

DOCUMENT RESUME

ED 324 039

JC 900 463

AUTHOR Tripe, Robert L. K.
TITLE Problem Solving and Writing: A Teaching/Learning Model for Computer Studies.
PUB DATE 23 May 90
NOTE 35p.; Paper presented at the International Conference on Teaching Excellence and Conference of Administrators (12th, Austin, TX, May 20-23, 1990).
PUB TYPE Speeches/Conference Papers (150) -- Reports - Descriptive (141)
EDRS PRICE MF01/PC02 Plus Postage.
DESCRIPTORS Communication Skills; Community Colleges; *Computer Science Education; *Curriculum Development; Job Skills; Peer Evaluation; *Problem Solving; Program Descriptions; Programing; Teacher Effectiveness; *Teaching Methods; *Technical Education; Two Year Colleges; *Writing Across the Curriculum; Writing Assignments
IDENTIFIERS Mohawk College of Applied Arts and Technology ON

ABSTRACT

In an attempt to combat the problem of poor writing and problem-solving skills among technical program graduates, Mohawk College of Applied Arts and Technology in Ontario, Canada began implementing changes in its Computer Studies curriculum. The first change was a greater focus on improving students' writing skills. Writing assignments were developed to be relevant to students' chosen field of study and future career, and were evaluated by both faculty and students. Another important change to the Computer Studies program was to augment the content-driven technical courses with the teaching of actual problem-solving skills. A model was developed for incorporating problem-solving strategies into the computer programming curriculum through a three-step process. The students: (1) define the nature of a programming problem in their own terms; (2) reformulate a problem and its proposed solution through a clear and unambiguous sequence of steps; and (3) translate the solution into a language (or "program code") which can be fed to the computer for action. This three-step model has been used in courses in Programming Methodology, Interactive Programming, and Computer Hardware Organization and Function. An appendix provides sample writing, problem-solving, and lab assignments for use in the computer studies classroom. (JMC)

* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *

ED324039

JC 900463

PROBLEM SOLVING AND WRITING:
A TEACHING/LEARNING MODEL FOR COMPUTER STUDIES

Robert L. K. Tripe, Professor
Computer Studies Department
Mohawk College Of Applied Arts & Technology

"PERMISSION TO REPRODUCE THIS
MATERIAL HAS BEEN GRANTED BY

R. L.K. Tripe

TO THE EDUCATIONAL RESOURCES
INFORMATION CENTER (ERIC)."

U.S. DEPARTMENT OF EDUCATION
Office of Educational Research and Improvement
EDUCATIONAL RESOURCES INFORMATION
CENTER (ERIC)

☐ This document has been reproduced as
received from the person or organization
originating it.
☒ Minor changes have been made to improve
reproduction quality.

• Points of view or opinions stated in this docu-
ment do not necessarily represent official
OERI position or policy.

Paper presented at the International Conference on Teaching Excellence
and Conference of Administrators (12th, Austin, Texas,
May 20-23 1990).

Problem Solving And Writing: A Teaching/Learning Model For Computer Studies

Robert L.K. Tripe, Professor
Computer Studies Department
Mohawk College Of Applied Arts & Technology
Hamilton, Ontario
L8N 2T2

Abstract

The future workplace requires a workforce able to adapt quickly and appropriately to changes brought about by both market and technological forces. Community colleges must re-focus their curricula to meet these demands by integrating the teaching of the skills necessary to enable graduates to succeed in this arena. It is my contention that a conscious effort to incorporate problem solving and writing skills development is the solution to the needs of our students and of the workplace. Beginning in 1986 within the Computer Studies Department of Mohawk College several courses were modified to develop teaching and learning methodologies in these areas. This discussion explains the rationale and implementation problems as well as illustrating solutions based on our experience to date.

Preamble

I would like to begin with a discussion of exactly what I perceive the problems to be in preparing our students for the workplace of the nineties and beyond. Before doing so, however, I feel I should clarify a few issues. Prior to joining the college I was a "practitioner" for ten years — a programmer/analyst with a research and development group. This proposal represents a distillation of ideas and conversations which have taken place with colleagues in many locales since I joined Mohawk College in 1979. The point is that I'm *not* an educator by profession other than in the sense of the user training and documentation necessary to implement systems that I developed. Secondly, my background in research and development has placed me in a good position to analyze the needs of students vis-à-vis communications, problem solving and research skills.

When proposing any new model, there is a need to point out the deficiencies with the existing model for the purpose of comparison. Unfortunately

this tends to leave an impression that there is nothing good about the existing system — this is definitely *not* the case with the Computer Studies Department at Mohawk College. Since its inception in 1967, the department has been a very successful example of a co-operative educational program. That our graduates are held in very high regard is evidenced by the fact that virtually all of them have jobs in the computing field within a few weeks of graduation. However, the advent of the micro-computer and associated technologies has created an unprecedented upheaval in the computing profession. Since the profession is well entrenched, it has a certain amount of inertia and consequently the changes haven't made themselves completely felt — yet! I feel that if we don't change our teaching strategies now that the employment figures for our graduates will begin to slip, beginning in the mid-nineties.

The Problem

Employer surveys and studies of the workplace have defined desirable employees as those who: (in order of priority)

- 1 ***have effective writing and oral communication skills***
- 2 ***can work independently and as part of a team***
- 3 ***are self-starters and life-long learners***
- 4 ***have technical knowledge at the job entry level***

Interestingly, and unfortunately, within technology we have tended to concentrate on teaching technical competence — the *lowest* priority skill identified! That this is the case shouldn't be all that surprising given the profiles of a typical community college professor. By and large, professors are either pure academics at the Ph D. level in their fields or experienced practitioners with a bachelor's degree — the author being a case in point. Putting aside the relative merits of the two profiles (*my position is a given*), neither of the two types has likely had more than a smattering of training as a *teacher*. As novice teachers facing the terrors of the classroom for the first time, we naturally fall into the trap of regurgitating facts simply because it is easier to do so — when you're up to your hips in alligators it's hard to remember that your objective was to drain the swamp. Thus, we begin our careers by teaching ***content*** instead of ***process*** skills, how to use

facts to solve problems. Of course, once in the groove it is extremely difficult to get out of it.

Teaching Technical Knowledge

As mentioned, this is the easiest type of teaching to do and, incidently, the easiest to evaluate:

"Just learn all this stuff and you will pass the course"

Perhaps the major difficulty in this regard lies in simply keeping up with technological change and ensuring that our offerings are relevant. This also provides us with an excuse for not changing our courses:

"I can't take any of the content out, in fact, I need more hours per week to put more content in to keep up with change!"

When pressed about whether or not we teach process skills, the reply is often that we **do**. In computer courses we use phrases like "analyze the problem", "sketch the logic with a flowchart to verify that it makes sense", "make sure you understand the problem before trying to solve it" and so forth. While this sounds terrific, we don't actually teach our students **how to do these things** — typically we do it for them in class and then turn them loose to do it on their own.

Consider just one of the phrases, "analyze the problem". What we mean is to break the problem down into its component parts, solve the resultant smaller and therefore simpler problems and then put them all back together to form a solution to the whole problem. The question I would ask you to ponder is whether or not you show your students *how* to break the problem down? How will they know that they have broken it down completely? How will they know whether the component parts are *really* part of the solution? What action should they take when they are stumped to begin with — they don't know where to start?

Teaching Students To Be Self-Starters And Life-Long Learners

This one is generally relegated to the bins labelled:

"They'll pick it up automatically while they're at the college"

"We're technology teachers, not psychologists!"

Even when the notion isn't pigeon-holed with the hope it will go away, we typically just pay lip service to it. For example, we force students "look up" the odd topic on their own by saying they're responsible for it because it's in the course outline. To drive the point home we tell them that they won't get the kind of "spoon feeding" in the workplace that they're getting at the college....hmmmm!

Teaching Independent Work And Team Work

We hope that we're giving our students the opportunity to learn these skills with assignments which are to be done as individuals and graded individually or as team assignments and graded collectively (all team members receive the same grade). By and large I believe we are doing a fairly good job on this one — there is, however, a caveat. Success presupposes that we have taught our students how to solve problems. If we have not then such assignments will be of little use because the students who are unable to solve the problem will simply copy a solution from someone who has. Students are pragmatic — in a course which focusses on content, copying will often be a successful strategy because the examination is based on the very assignments for which they have copied a solution to memorize.

To be useful, assignments must be achievable, challenging and perceived by the students as being relevant in the workplace. Assignments which are perceived by students as "course outline sensitive" are deadly. These are assignments "dreamed up" by a teacher so that it covers all of the course outline components within a section of the course but isn't particularly practical. Assignments which can be done "the night before" are worthless and perceived to be so by students. Assignments which are so tricky that only a few students can figure them out are also of doubtful worth — they aren't perceived to be "real world" examples and are accorded little respect and effort. Tricky assignments also raise the frustration and hostility levels of the students which will make it tougher to teach them anything!

A good assignment should take 10 - 15 hours to complete and should provide the satisfaction of achievement upon completion. Such assignments must be relevant to the workplace (point out and discuss the relevance) and wherever possible should relate to other courses. Lastly, good assignments are those which challenge students to use knowledge and skills from as many areas as possible — cross-course and cross-discipline.

Teaching Communications Skills

In terms of teaching this is probably one of the weakest areas faced by the community of "technocrats". I don't think this should surprise anyone because, by and large, we "technocrats" were never required to do much in the way of written or oral communication. Ironically, our own backgrounds are the same as those we're providing to our students: take a few "English" courses because they're part of the curriculum.

In short, we delegate the teaching of writing skills to a language studies (or equivalent "English") department for one or two semesters. The instructions we give to such departments are along the lines of teaching our students business communications, resume writing and the like — rather nebulous to say the least! While these types of writing are important, they don't begin to cover the gamut of writing required in the workplace. Our students need to learn how to write proposals, manuals, user instructions, memos, explanations and reports to name a few.

The Answer: Some Thoughts

Perhaps the biggest difficulty we face lies in realizing that we even have a problem! The demand for our graduates is high, there are more co-op jobs available for the students' work terms than there are students....so *what's the problem?* As I mentioned at the beginning, the workplace is changing and we must change with it or perish — our curriculum was designed for the seventies and eighties, not the nineties. Another factor is that the student of today is different from the student of 10 - 15 years ago in terms of entry-level skills (coming in our doors from high school). We are also facing a major re-training task to upgrade an existing workforce. Study after study has shown that these students have weak process and communication skills (low scores on math inventory and English

proficiency tests for example). Unhappily, studies have also shown that these are the very skills that are necessary to survive in a workplace characterized by rapid change and a consequent need for learning and the ability to learn.

It is easy to point the finger elsewhere:

We haven't changed our courses or the way they are taught so it isn't our fault if students perform poorly in our classes.

Blame the students themselves for their lack of skills and righteously proclaim that we won't lower our standards so its up to the students to somehow "pick up whatever they need".

Whomever taught the students last (the public school system) is to blame for the students' lack of skills.

It's not our job to remediate so we shouldn't accept students who don't have a reasonable chance to succeed.

While much of the foregoing may be true, the fact is that our student base is not likely to change and, if we don't address the problem then who will? Remember that employers blame any perceived deficiencies on the institution which taught their employees last — that's US, I believe. As individual departments it is up to us to teach students how to acquire the skills necessary to succeed and grow in the workplace.

Relying on other departments to deliver courses in writing or critical thinking (problem solving) simply won't work for the skills we're trying to teach — individual courses in specific subject areas are taken, passed and forgotten by students, another checkmark against the requirements to graduate. While we know that this is true (ask any student), we continue to let others design and deliver such courses. It is my opinion that the ONLY way to address the need for the kinds of skills employers and the workplace demand is to integrate the teaching into all of the courses that we (within a department) teach. To do otherwise is too much of a hit or miss proposition.

Of course, convincing all of the faculty within a department to accept and implement this notion is tough sledding — convincing an entire institution is, shall I say, *challenging*? Corollaries to this problem are the design of courses, delivery methods and faculty training along with overcoming faculty apprehension and inertia.

What Have We Done?

To begin with, *we have begun the campaign* — demonstrate by *doing* and the world will beat a path to your door. In 1986, one of my colleagues (Ted Thorpe) and I determined to consciously try to change a few courses to see if it was possible to implement some of our ideas in a practical sense. The remainder of this paper discusses the rationale and provides some examples of what we have done to date, followed by some thoughts on the future.

Writing Across The Curriculum

In the past we have asked the Language Studies Department to deliver courses which emphasize writing and reporting skills; students are expected to use these skills for resumes, in systems analysis courses and a major work term report. That our students cannot write despite these courses is obvious from the written work submitted (systems analysis and work term reports). The problem does not lie with the Language Studies Department — they fulfill their mandate very well indeed. The problem lies with *our* department (Computer Studies). From the perspective of a student in our department, out of 38 courses taken towards a diploma, **only 6** focus on communications skills to any great degree. If you were a student, do you suppose that you would assume that these skills had any particular importance in your field?

Based on employer studies and our curriculum at the time, it is perhaps not surprising that writing was one of the top priority items on the list of areas for change. The question to be answered, of course, was "How were we to introduce writing assignments within what is basically a technology program?" At first glance this would seem to be difficult — in practice this has not been the case. In fact, we are experiencing the opposite problem, namely that we can come up with more writing assignments than our students and our curriculum have time for! Appendix A contains several examples of writing assignments that we have used over the past few years in a number of different courses. I will leave it to the reader to peruse the examples to get a feeling for their "flavour". In passing, I should mention that the assignment sheets themselves should be produced with some care towards their appearance; they serve as a model of the "look" you expect the students to strive for. The wide availability of word processing programs means that you can reasonably expect students to produce good looking documents which have been carefully checked for spelling.

Incidentally, the term which describes the kind of writing we're asking our students to do is often called "Writing Across The Curriculum". The benefits to be gained from writing are clear and have been discussed widely at symposia and conferences for the past few years. I won't belabour the point other than to suggest that you to study Bloom's Taxonomy — writing helps all of the higher levels of learning such as Analysis and Synthesis as well as the lower levels such as Knowledge.

Before I began this project, the reaction to the proposal was almost universally negative — to wit, the students can't write and won't write. I have found, however, that students not only can write but enjoy it and improve dramatically when given the chance to do so. Some care must be taken with your own expectations because it is easy to be negative in your evaluation of their work. For many of our students, this will be the *first writing of any substance that they have done*. It is *unrealistic* to expect Shakespeare, it is *realistic* to expect clear, logical, well presented material. I have some further thoughts on the matter of evaluation below but, first, some notions on how to create writing assignments.

Writing: Relevance

Writing assignments must be *relevant* to the students' chosen field and it is up to the instructor to point out the relevancy. In effect what you are doing here is *selling* the idea that writing is important. My idea is that you simply study the kinds of writing the students will be expected to do when they join the workplace and then have them practice. Employers are a tremendous asset in this regard and will often provide you with actual examples (which go a long way towards convincing students of the relevance). One of my favourite dialogues in class is as follows:

How many of you (students) have read computer manuals?

How many of you have found them to be useful, helpful, clear, easy to use and so forth? (Usually don't get too many positive replies on this one!)

Why do you (students) suppose that these manuals are so terrible? (Don't usually get an answer on this one but sometimes generates some discussion)

My answer... *Because it is people like YOU (students) who wrote them! (Enough said)*

Once students are convinced that writing will be an important element in their careers, they will write. Some care must be taken to ensure that the objectives of the written assignments are clearly stated — although it is interesting to give

them assignments which require that they do some initial research to do this clarification on their own, providing that it is clear that this is one of the objectives! If they are given writing assignments in *all* of their courses then they will be well prepared for the workplace. Furthermore, the process of writing will help them to *learn on their own* by developing research skills.

Writing: Evaluation

Now to the thorny issue of evaluating students' work. A common roadblock thrown up by fellow professors is apprehension that since they are not trained as "English" professors, how can they evaluate written work? I usually reply with a series of questions:

Can you read a report and indicate places where the points are unclear or contradictory?

Can you tell that a passage or series of passages aren't logical or contain questionable facts?

Can you decide whether a written argument makes sense or that a conclusion is justified?

Can you point out sentences which are poorly structured or awkwardly phrased?

Can you show places where passages should have been broken up into two or more paragraphs?

All of the foregoing are things which any of us ought to be able to do — it isn't necessary to get into subordinate clauses and prepositions to tell whether a piece of writing is bad. I've used criteria such as those above for the past few years and found them to be more than adequate and quantifiable (Appendix A, Report Format). Another interesting technique which I've used to good effect is to have students evaluate each other's work and award a grade as part of the overall grade for the assignment. Again, the naysayers warned that I would encounter resentment and reluctance and, again, they have been proven wrong. Students not only do the evaluation but often write comments to their peers — both complementary and negative ones! In this vein, I often *encourage* students to proofread each others work since this is a common practice in the workplace.

As a final note with respect to evaluation, consider that the students will have spent a good deal of time doing the work depending on the size of the report. It is therefore only fair that the evaluation be done carefully and thoughtfully as well. I include a section on the report cover page on which I write

my reaction to the work along with suggestions for improvement (if necessary). When I hand back the work, I notice that students immediately read my comments. I also hold back the best examples to hand out last and publicly complement these students' work and suggest that others read through them. Incidentally, I complement *all* of the students publicly in the process of commenting on the work in general — "a spoonful of sugar to help the medicine go down".

Problem Solving

For some time, now, my colleagues and I have been puzzled as to why some students (very few) take to computer programming and related courses while others struggle to succeed. The former group just seem to grasp the notions and run — we don't see them for dust. Most of the second group ultimately succeed to a lesser degree and some within this group don't ever catch on and fail. The solution to this problem eluded us for some time but had to be solved if we were to prepare our students to meet the demands of the workplace. By a process of elimination we finally decided that our students simply ***didn't know how to solve problems***. This, of course led to a secondary puzzle, namely, why can't they solve problems? Problem solving is a pervasive activity in virtually all aspects of the students' lives and yet they can't solve computer related problems.

My colleague, Ted Thorpe, and I examined this puzzle and, after much discussion, determined that there were two problems to be solved:

Content-driven courses and teaching.

Lack of integrated problem solving methods.

Content-Driven Courses

Most of our courses fall into this category, we create courses and outlines which contain lists of *things* students must learn to pass the course. The delivery of the course then follows the outline in lock-step fashion allowing the instructor to "tick off" the objectives as they are covered. Such objectives are far too often of the "Upon completion of this objective the student must demonstrate how to use...." variety. An analysis of these type of objectives shows that they fall into the Knowledge category of Bloom's Taxonomy, in other words, the lowest level of learning. Ironically, this is also the lowest priority identified by employers as desirable

employees skills! And, of course, these are also the *easiest* types of objectives to teach to.

In the case of a computer programming course, the objectives tend to fall into a list of language elements and syntax along with short examples illustrating how they are to be used. Examples with more "meat" in them can't be taught because they would take too much time away from the list of *things* to be included. In the process of teaching programming, we use phrases such as "analyze the problem", "depict the logic using flowcharts" and then "code the solution". Unfortunately, we **never tell them how** to "analyze" anything. We simply assume that by looking at our examples and writing programs that they'll somehow learn to do so.

Problem Solving

Granting that our students don't apply the requisite problem solving skills raises a question: how do we teach the skills while retaining the integrity of our courses? Before tackling the problem, however, we had to determine exactly what skills the students needed to *apply* — we assumed that they already *had the skills*. We felt that if we could re-structure our courses to put more emphasis on problem solving that course content shouldn't suffer because we could move more responsibility for learning to the students. By doing this, we would also be addressing the needs of employers and the workplace.

It became quickly apparent that if we were to adopt a problem solving approach that we would need new texts to support the notion. Unfortunately, while there is material on problem solving in general and sections of texts in other disciplines (according to colleagues), there is nothing which relates to computing. My colleague, Ted Thorpe, and I are in the process of rectifying this deficiency and hope to have a text written by sometime in 1991.

Basically, we feel that, among other things, students need a model of some kind to work with. Following is an excerpt from our writings to date which discusses a three step solution model which is adaptable to almost any discipline.

Three Step Solution [®]

When you are faced with a problem to be solved with a computer, there are three phases or steps to follow:

1. Teach yourself what the problem is in terms you understand.
2. Re-formulate the problem as a clear and unambiguous sequence of steps.
3. Transfer the solution to the computer.

Step 1

The objective is to "make sense" of the problem in terms you understand. You must try to view the problem from as many frames of reference as possible including your own and those of others involved in the problem or project. You are also gathering and sifting facts and their inter-relationships which result from these various frames of reference. Ultimately, you will eliminate everything from this cornucopia of information except what is relevant to the problem solution. This is a step requiring effective questioning techniques and a high degree of skill in their application — think of yourself as an investigative reporter trying to uncover a government scandal!

While we don't want to over-emphasize any particular step, this first phase seems to be one of the toughest for novices. Typically, the problem is examined superficially and from a singular, very narrow point of view; a novice then quickly proceeds towards a solution suggested by this "analysis". Alternatively, the problem is perceived to be "too difficult" because no frame of reference can be readily found and the novice gives up. *Our* objective in this text is to help you to overcome these hurdles to successful program design.

Step 2

You are preparing to explain the solution to a computer; this means that you will have to establish a common frame of reference with the computer. One reason that this step is so difficult is because we are used to dealing with *people* and people draw from a rich store of frames of reference and are able to shift from frame to frame until

Copyright © 1990 by E. Thorpe, R.L.K. Tripe

a close match is found. Computers, on the other hand, have only a limited store of very rudimentary frames of reference and can't shift amongst them. You need to simplify your explanation to the level of your audience, but, since you are accustomed to working with people you assume frames of reference and associated knowledge far beyond anything possible with current computers.

Novice programmers have a great deal of trouble in appreciating how simplistic the computer's frames of reference and knowledge bases really are and thus fail in this second step. Solutions must be broken down into a well-defined, robust sequence of simple operations before proceeding to the last step.

Step 3

The third step should be the easiest one to do provided that the first two have been completed. It is essentially a mechanical operation involving the *translation* of step two into terms or a *language* (the program code) which you can feed into the computer for action. Step three is what many people refer to as *programming* and novices tend to proceed to this step first, for the most part bypassing steps one and two. Inevitably, the results aren't as expected — the computer follows the sequence you've fed in and doesn't produce the desired effect.

In an attempt to discover why the *program* doesn't work, our novice may back up and skim through steps one and two. The resultant "new insights" are used to modify the *program* and try it again. This circular process may ultimately lead to success of a sort providing that the problem to be solved is simple enough; the process will not work in situations of any complexity. Even more distressing is the case where novices keep flogging away at step three *without* going back through steps one and two:



Never try to teach a pig to sing,
it wastes your time and it
annoys the pig.

Novice programmers often become frustrated and angry at this stage because they don't experience success and they can't understand why. Such novices have not yet realized that *programming* involves all three steps. We prefer to call step three the *coding* step while a *program* is

a composite of all three steps, not just the last one. Novices cling tenaciously to their "step three only" solutions in part because they have invested a fair amount of time

and effort in the solution. Experience tells them that spending a lot of time on a project yields results and so they continue to put in time. Any suggestion to a novice that they are wasting their time is not received with any measurable degree of gratitude.

Discussion: The Three Steps

While we don't pretend to have all of the answers to how people solve problems, we do know that they ought to be able to write computer programs. The big question to be answered is how do we teach you to write programs? As with any other learning situation, we can only point out some ideas and techniques which seem to work and the kinds of skills you must bring to bear — the rest is up to you.

As mentioned, making a computer do what you want it to (via a computer program) involves three steps. Attempts at completing the third step before completing the first two generally result in computer programs which don't work (or don't perform as expected). A reason for this lies in determining exactly when you have completed the first two — not an easy task as we shall see.

Our goal, then, is to explore these three steps and show you how to bring your skills to bear on the problem. Note that we are emphasizing the fact that you *already have the skills*, it's a question of recognizing them and consciously applying them.

People are natural problem solvers, perhaps because the ability is a survival tool; we aren't as strong or fast as many other species and so have developed other survival techniques. While many of the so-called "lower animals" must also solve problems to survive, it appears that they do so by slavishly following a relatively few learned patterns. When placed in situations which require the generation of entirely new patterns, animals either fail completely or at best perform poorly. Humans, on the other hand, *can* generate solutions by searching through accumulated knowledge and re-arranging patterns to create new ones. (Whether this model is technically correct or not, we find it is a useful viewpoint when honing your skills).

What we are attempting to do with this book is to help you to consciously direct your skills towards the solution of problems to be solved using a computer. Before proceeding, however, we must first examine how problems are solved with computers as opposed to the way people solve problems and then we will illustrate some of the difficulties involved in the first two steps of our three step solution.

Revamped Courses

As a department, we are progressing towards the implementation of a problem solving approach within our curriculum — courses which we teach to our students. These are the only ones over which we have direct control; we can only "suggest" to the other departments involved that our approach is better than theirs — a tricky process of diplomacy to be sure! The following are brief descriptions of a few courses which are either new or which we have changed along with a few thoughts on each — space does not permit a full discussion. We have worked on 8 or 9 courses so far, so view this as an "in progress" report.

Programming Methodology

Having agreed that our students couldn't solve problems we decided that we should address the issue in the first semester with a "bellweather" course. The design criteria for the course were:

that it address problem solving from a computer programming perspective

that it teach students techniques which they would be expected to use throughout the rest of the program of studies and their careers

Since this course would teach students how to *design* computer programs we decided that there should be no actual program coding per se (step 3 above). We felt that if we were to "muddy" the water with coding that the course would tend to fall back into a "content only" base and undermine its purpose. This decision also meant that we would not have any programming in our program of studies until the second semester — a radical departure from conventional wisdom! Our reasoning was that if students didn't know how to solve problems (steps 1 and 2 above) then they certainly couldn't be expected to code them. In other words, they would start their program of studies by developing bad habits in one the very first courses they took.

Basically the entire course focusses on steps 1 and 2 above. Students are presented with problems which they must analyze by breaking them down into components, viewing problems from different points of view, trying to decide whether solutions are complete, determining what further information (if any) is needed and so forth — basically step 1. This is accomplished by spending a good deal of time in class using problem solving techniques to solve problems. Students

then proceed to define types of logical structures, rules of modularity, depicting logic and dependencies with flowcharts and hierarchy diagrams — step 2. Throughout the course students must write short descriptions, problem statement re-formulations, examine and critique problem solutions and statements to name a few. At present, 4 out of the 15 weeks are devoted to some introductory level coding which we hope can be expunged in the near future.

As might be expected, the course is not without its flaws, however, considering how far we've come, I feel that it has been successful. Several professors have taught the course (there are typically six class sections of 40 students) since its inception in the fall of 1988 and may now be safely classed as "converts". One of the major difficulties has been the lack of a text, a problem which we hope to remedy, as mentioned. In the interim, however, I prepared a short document entitled "Creative Problem Solving For Programmers" for use in the course.

Interactive Programming

This is a third semester course, originally entitled "BASIC Programming", now changed to "Interactive Programming". The course taught students how to write programs using the BASIC language and was a typical "programming" course — language syntax, short examples, several programming assignments. After a major re-design, it is a course which teaches interactive programming techniques using the BASIC language as the implementation vehicle. Students are expected to research and familiarize themselves with most of the language and syntax necessary to code solutions. Only 20% of the class time is devoted to discussing the language itself, the rest of the time is spent exploring solutions to problems.

In an effort to make this course exemplify the "real" world, only one programming problem is tackled throughout the semester. It is our contention that this is superior to the notion of working with many smaller, unrelated problems and assignments — they confuse the issue. We've used an airline reservation system and an integrated sales/inventory system so far with a student information system in the works for the fall of 1990. These problems are far beyond a complete solution but the fundamental parts can be dealt with. Problems such as these illustrate the fact that complex problems simply can't be solved unless the techniques learned in Programming Methodology are employed. Class time is spent analyzing the problem as a whole along with various sub-problems as they come up. Students design partial solutions and attempt to implement them; as more sophisticated

techniques are introduced they replace earlier, usually more cumbersome solutions.

The underlying idea is to let students try to solve the problem one way and then help them to discover a better one by introducing a new thought, element of the language or, perhaps, information about how the computer hardware works. By and large, it is easier to let students "hang themselves" on a poor solution and then devise a simpler, more elegant one because it will be a lesson well learned. Throughout the course, students are constantly pushed to explore the ramifications of solutions and compare alternate solutions before implementation (cycling through steps 1 and 2).

Student reaction to this course varies from the thrill of discovery to angry frustration (often in the same student!). We're not surprised by the anger because these students, for the most part, have never been given this much responsibility for their own learning before; they complain that we are supposed to "teach" them the language. Secondly, they are still trying to go to step 3 before traversing steps 1 and 2 — the problems we're attacking cannot be solved this way and frustration sets in. Thirdly, many of the students are convinced that they are not successful because they don't know the language when, in fact, they haven't solved the problem. Failing to understand the computer language syntax and usage is rarely a major problem.

Computer Hardware Organization And Function

This is a new course (fall 1990), designed by Ted Thorpe from the ground up to incorporate as much of our thinking as possible. The potential for this to be of the "lecture/fact-based" variety was great so Ted skipped out of the groove by deciding that it would be primarily a lab-based course using microcomputers. A constraint of the computer labs is that they are used to teach software packages to a broad spectrum of students, not just those in computer studies. By taking this approach, he was forced to determine a way in which students could discover how computers worked without actually taking them apart. A further problem was how to have them do this "discovering" without teaching them machine language programming.

The solution uses a software package, **TURBO DEBUGGER AND TOOLS** from Borland. This package allows students to load programs into the computer and then examine them at virtually all levels from source code to single step execution at the machine level. In the lab students are expected to answer questions and

derive questions on their own. Small groups will also be expected to present findings to the class as a whole during the classroom portion of the course. I've included a sample of the lab work to be done in Appendix A (course code is CO343). Of course I can't provide any student reactions to this one *but* I expect that it will be excellent and should address all of the skills mentioned by employers.

The Bottom Line

The battle has been enjoined! From a small beginning we have begun to make changes to meet the needs of employers and the workplace. At this stage, it is very difficult to provide quantifiable results if, in fact, we can ever quantify results. We're aiming at a moving target and therefore won't likely have a baseline against which to measure other than the continuing success of our students. One thought in this regard, however, might be to begin to track both our graduates and their employers find out whether we are meeting market demands.

Student reactions are particularly difficult to measure in any meaningful way other than through their willingness to do the things we ask them to do. Thus far, they have reacted positively as long as we have taken the time to "sell the notions". They are being challenged in ways most of them haven't seen before and, unless they are convinced of the worth of the methodologies, they give up. In general, students seem to be happy with the new courses and are succeeding in them — failure rates haven't increased, for example. Because many of the courses are such radical departures from previous offerings, it isn't valid to compare grades. The attrition rate hasn't changed appreciably either so we can't use it as a measure although it is an area which we are examining.

The reaction of fellow professors has been positive once the initial apprehension and skepticism was overcome. In general I would say that the level of enthusiasm and innovation has increased rather markedly. We seem to spend much more time discussing exactly what skills our students need and ways to teach these so-called "soft skills". This is a shift from previous arguments over what topics could and could not be included due to time constraints, complexity and so on. We have begun to move away from the slavish adherence to content teaching and evaluation methods. The atmosphere with respect to evaluation is changing to examine more realistic evaluations which reflect the workplace, in other words, **demonstrate skills and know facts**. We're good at the latter method but still learning the former.

As final word, consider the benefits to implementing a writing and problem solving approach across the curriculum. Aside from meeting current market demands we can have a more "reactive" curriculum. If our students are effective problem solvers and communicators then "fast track" reactions to market needs are more easily implemented for two reasons:

Students are better able to take responsibility for their own learning and can succeed in courses which haven't been as fully developed as we'd like.

Professors are better able to quickly develop and deliver new courses using flexible delivery mechanisms in an atmosphere of innovation.

Demands on the community colleges has never been greater and funding will continue to be a battle. We must train students fresh out of high school as well as those already in the workforce. It is my view that by integrating the skills mentioned that we can meet the needs of the present and future workplace. I also feel that it is not a question of whether or not to do so but, rather, a question of *HOW*. To paraphrase:

If we aren't part of the solution then we are part of the problem.

APPENDIX A

Report Format

Each report submitted must be prepared and printed using a word processing program. The report must discuss the objectives as stated on the cover page. The cover page, with appropriate sections filled out by the student, must be stapled to the front of the report.

Where reference material has been used (texts, periodicals, interviews and so forth), they must be listed in a bibliography on a separate page at the end of the report. Each such item must include title, author, publisher and year and page numbers as appropriate. Use the example below as a guide. Material quoted verbatim must be in "quotes" and a subscript corresponding to the item number in the bibliography placed at the end of the passage. Material which has been paraphrased must also be subscripted. Since a major purpose of the reports is to give you practice in expressing yourself coherently, I would suggest that quotes and paraphrasing be kept to a minimum.

Examples:

1. SHOCH, J.F.: "Performance of an Ethernet Local Area Network", Simon and Schuster, 1981.

2. Steel, K.: "Telephone Technology and LANs", Computer News, vol 22, p. 67-84, May 1985.

The body of the report should follow the guidelines below as to order and content (Figure 1). Use underlined headings for each section and topic. Diagrams must be neatly drawn in pencil or ink using a straight edge, templates, compass etc. and must be titled. Diagrams must be numbered consecutively and referenced in the text as Figure 1, Figure 2 and so on.

- Cover Page
- Introduction
- Topic 1
- Topic 2
-
- Conclusion
- Figures
- Bibliography

Figure 1. Report Layout

Introduction

A brief overview or summary of the report purpose, content and conclusions.

Topics

These will correspond to the objectives although not necessarily in the same order and, where provided, the list of questions to be answered.

The topics will consist of descriptions and/or explanations of the topics in your own words. If it seems logical to do so, several topics may be combined under one general heading.

Conclusion

Depending on the objectives, a conclusion may or may not be required. If it is required, make sure that the topics contain sufficient justification to back up your conclusion.

Figures

These may either be included directly in the body of your report or collected at the end prior to the bibliography. If collected at the end, you may put more than one figure on each page as you see fit. For the sake of clarity (and that's why you're using figures!), don't try to cram too many on one page.

Bibliography

Content as already stated, must be on a separate page.

Marking Scheme: Reports

The letter codes below will be used to indicate problem areas within your report; your instructor will deduct approximately .1 for each occurrence.

Layout

- Q** Quotation not specified
- T** Titles etc.
- B** Bibliography

Content

- L** Not presented in logical sequence, does not make sense
- U** Unclear
- F** Questionable facts
- C** Contradiction
- M** Missing description or explanation
- Nr** Not relevant

Readability

- Ss** Sentence structure
- A** Awkward phrasing
- P** Paragraph
- S** Spelling/Proofreading

Computer Studies Department

Data Communications Report

Program:	Data Processing
Course:	Data Communications
Code:	CO441
Module #/Unit #:	1.1 - 4.1.2
Title:	Report 1: Wide Area Networks Overview

Objectives:

Your data network consulting company is making a presentation to a potential client at the client's next board of directors meeting. The client wants to move towards distributed processing but needs board approval. This presentation is to be a briefing session in which you are providing an overview of some computer network concepts as they relate to common carrier transmission facilities. A major concern is that you **avoid jargon** because the board members have little technical background and would be annoyed by too much "tech talk". The presentation is to be brief and this written report is to be provide to board members to take away from the meeting.

Specifically, the client wants you to:

- discuss fundamental WAN concepts
- highlight the OSI model
- introduce transmission basics including serial, asynchronous and synchronous transmission
- explain the use of common carrier voice networks as compared to the digital networks
- explain how errors can occur and how they may be detected
- discuss the gains in efficiency possible through the use of multiplexors
- wind up with a quick note on FAX machines

Remember that diagrams are invaluable tools!

Special Instructions:

Must be produced with a word processing program.

Must be **eight to ten pages, double spaced**.

Consultant's report to a company president.

Materials:	1. Business Data Communications, David Stamper
	2. Data Communications Student Workbook, R.L.K. Tripe
	3. Class discussion

Evaluation:	Layout:	1.5
	Readability:	3.5
	Content:	5.0
	Total Mark:	/ 10.0

Evaluator Comments:

Student name:	Class:
Instructor:	Due Date:

Computer Studies Department

Data Communications Report

Program:	Data Processing
Course:	Data Communications
Code:	CO441
Module #/Unit #:	4.1.3 - 8
Title:	Report 2: Data Link Control & Message Delivery

Objectives:

Assume you work for one of the common carriers as a network analyst. A friend of yours is a systems analyst who works for a large insurance company and has been given the job of designing a computer network for the company. Your friend is a novice with regards to networks other than some fundamental knowledge and basically needs a summary of messaging from the front-end processor out as a first step.

In particular, your friend doesn't quite understand the whole notion of messaging within a network. For example, how are messages exchanged, how do you know who messages are for, how do messages travel through the network, what happens with large data blocks etc. Errors are a further major concern; transactions from remote ter-

minals or other computers have to be error free because finding them once they have been posted is next to impossible. Secondly, transactions have to be processed in the order sent so the question is "How do you know the order and can messages arrive out of order?". Another possibility is the use of a packet switched network so a brief explanation might be necessary.

Two final points need to be addressed. The transactions are very sensitive and some form of security is needed so your friend wants you to suggest some ideas. A fair proportion of the data sent is to and from terminals and contains repetitive strings - are there any ways to make this transmission more efficient?

Special Instructions:

Must be produced with a word processing program.

Must be eight to ten pages, double spaced.

Directed to a fellow systems analyst

Materials:

1. Business Data Communications, D. Stamper
2. Data Communications Student Workbook
3. Class discussion

Evaluation:	Layout:	1.0
	Readability:	2.5
	Content:	4.5
Peer evaluator: _____	Peer Evaluation:	2.0
	Total Mark:	/ 10.0

Evaluator Comments:

Student name:

Class:

Instructor:

Due Date:

Computer Studies Department

Operating Systems & Data Structures Report

Program:	Computer Systems Technology
Course:	Operating Systems & Data Structures
Code:	CO252
Module #/Unit #:	2
Title:	Report 1: Primitive Data Types

Objectives:

Given a set of data values, you are to create diagrams showing the exact binary representations of these data in each specified data type. Accompany the diagrams with brief explanations of how the reader can interpret the diagrams.

Given two sets of 32 binary bits, interpret them as each specified data type. The data to be used is on the reverse side of this page.

Objectives summary:

- to gain practice in representing data using various data types
- to observe the effects of proper and improper data type conversion

Problem data:

- show the following as packed decimal with 5 digits of precision: +21.78, -0.349, +2.78695
- show the following data as floating point having a signed 8 bit exponent and signed 24 bit mantissa: +16383.0, -16.251953125
- show the following as signed integer having 15 bits of magnitude: -10256, +7150
- show the following data as character using both ASCII and EBCDIC character codes: "Bob Smith is 18."
- (see reverse side for the rest)

Special Instructions:

Must be produced with a word processing program.

Write as a tutorial for a classmate.

Must be no more than three pages, double spaced.

Materials:	1. Class notes
	2. Any reference material you can find
	3.
Evaluation:	Layout: 5.0
	Readability: 5.0
	Accuracy: 5.0
Peer evaluator	Peer Evaluation: 5.0
	Total Mark: /20.0

Evaluator Comments:

Student name:	Class:
Instructor:	Due Date:

Computer Studies Department

Interactive Programming Report

Program:	Computer Systems Technology
Course:	Interactive Programming Programming
Code:	CO355
Module #/Unit #:	1.1 - 1.3, 2.2.1 - 2.3.5
Title:	Report 2: Screen Display Modules - Functionality

Objectives:

You are to write a two - three page report describing the functionality required for the display of user data entry forms. In doing this task, try to identify, from the user's viewpoint, what the screen must look like and the operations that will be performed during *data entry and editing*.

From this analysis state in terms of display *functionality* what a program would have to do to cause the display to behave in the manner expected by the user. Finally, attempt to identify a list of modules and the parameters needed by them that you would write in order to meet the user's display processing needs.

Write the report from the position of a Systems Analyst making a design recommendation to the design team.

Use an airline ticket (attached) as a sample data entry document. In summary you are to:

- identify a set of functional modules for the display of data entry forms
- define a *generalized* set of external sub-program modules to implement the necessary display functions
- gain an understanding of data entry forms from the viewpoint of the user, programmer and analyst

Special Instructions:

Must be produced with a word processing program.

Keep the user in mind!

Must be no more than three pages, double spaced.

Materials:	1. VAX-11 BASIC User's Guide
	2. VAX-11 BASIC Language Reference Manual
	3. VAX BASIC HELP System

Evaluation:	Layout:	2.0
	Readability:	2.0
	Content:	3.0
Peer evaluator: _____	Peer Evaluation:	3.0
	Total Mark:	/10.0

Evaluator Comments:

Student name:	Class:
---------------	--------

Instructor:	Due Date:
-------------	-----------

Computer Studies Department

Interactive Programming Report

Program:	Computer Systems Technology
Course:	Interactive Programming Programming
Code:	CO355
Module #/Unit #:	All
Title:	Report 4: Problem Analysis

Objectives:

You are to write a two - three page report which firstly analyzes the difference (or similarities!) between the way you *planned* to implement your form display and data entry as expressed in reports 2 and 3 and the way you *actually* implemented it in assignments 2 and 3. Identify similarities, differences and omissions and try to explain why they exist.

Secondly, of the eight problem solving skills listed in the note "Gaps Between the Problem Solving Skills of Novices and Experts", evaluate your level of skill as you believe it is as present (do not write this complete evaluation). In the report:

Identify the *three* skills in which you believe you have made the *greatest* progress and explain how you know progress has been made.

Identify which skill you must work hardest to develop further and explain how you came to this conclusion.

Special Instructions:

Must be produced with a word processing program.

Be honest in your evaluation.

Must be no more than three pages, double spaced.

Materials:

1. Your Reports #2 and #3
2. Your Program Assignments #2 and #3
3. "Gaps Between the Problem Solving Skills of Novices and Experts"

Evaluation:

Layout:	1.0
Readability:	3.5
Content:	5.5
Total Mark:	/10.0

Evaluator Comments:

Student name:

Class:

Instructor:

Due Date:

Computer Studies Department

Assembler Programming Report

Program:	Computer Systems Technology
Course:	Assembler Programming
Code:	CO550
Module #/Unit #:	2.3, 3.2.1 - 3.3.5
Title:	Report 3: Data storage definitions, Packed decimal operations

Objectives:

Seemingly everyone complains about poorly written and badly presented computer manuals. This is your chance to show the world how such documents ought to be done!

Devise a suitable layout for an **Assembler Programmer Reference Manual**; this format will be used for future work so take care with the design. Since this is a programmer's reference manual, you must include the instruction formats for the assembler language as well as an explanation of how the instruction works. You are working on **Section 4, sub-section 1, the Decimal Arithmetic** section.

Briefly *introduce packed decimal* concepts (external vs. internal data representation) and then explain:

- data storage instructions *DC* and *DS* as they apply to decimal processing
- *PACK, UNPK*
- *AP*
- *SP*
- *MVC*

Special Instructions:

Must be produced with a word processing program.

Devise a suitable Assembler Manual format

Must be no more than four pages, double spaced.

Materials:	1. Assembly Language Programming, Kudlick
	2. Work term experience
	3. Class discussion

Evaluation:	Layout:	2.0
	Readability:	2.0
	Content:	3.0
Peer evaluator: _____	Peer Evaluation:	3.0
	Total Mark:	/10.0

Evaluator Comments:

Student name: _____ Class: _____

Instructor: _____ Due Date: _____

Lab Familiarization**Objectives:**

To develop a set of basic skills in the use of the debugger to view a program, to execute a program step by step and to display or modify program variables.

References:

- TURBO Debugger User's Guide, Chapters 2 and 3

Submission:

at the end of the lab

- a printed copy of your pre-lab plan
- printed copy of the file TCDEMO.C
- printed copy of the debugger log file created during the lab (TCDEMO.LOG)

before the start of the next lab

- a printed copy of the answers to the questions on this lab and the questions in the log

Assessment:

_____	Completeness	_____	/5
_____	Clarity	_____	/5
_____	Knowledge	_____	/5
_____	Planning	_____	/5
_____	Total	_____	/20

Comments:

_____**Identification:**

NAME _____

STUDENT NUMBER _____

NAME _____

STUDENT NUMBER _____

SUBMITTED TO: E. THORPE

DATE _____

Lab Familiarization - Detail

Tasks:

- 1 Before coming to the lab prepare a brief plan outlining what you propose to do during the lab.
- 2 Copy from the system to your floppy disk all files named TCDEMO (suggest TCDEMO.* as file specification for the copy).
- 3 Use the debugger to examine and execute the program TCDEMO.EXE following the guide given in chapter 3 of the reference.
- 4 Repeat the debugging exercise using the logging capability of the debugger to log the contents of at least 4 different windows viewed during the exercise in chapter 3. Use the Add feature of the logging window to include explanations of the windows in the log file. Name the log file TCDEMO.LOG.
- 5 Include in your log file at least 4 questions about the displays you saw or the debugger commands you used the answers to which you feel will help you use the debugger more effectively.

Questions:

- 1 Explain how you were able to determine which line of the program would be executed next as you stepped the program from start to finish?
- 2 Identify two debugger features not used in the chapter 3 exercise which you think would be useful to examine the workings of the computer in more detail and explain why you made your choices.
- 3 How many files do you have with the name TCDEMO? How was each of the files created? Which of them are printable? What is the importance of the information in each?

Topic:

Purpose and use of the debugger.

Suggested Subjects:

Based upon what you observed in lab 0 through the debugger draw diagrams depicting your views of what the computer looks like. Be prepared to explain how your observations support your views.

Discuss the kinds of things the computer seemed to be able to do in a single execution step. Try to explain the different interpretation of 'step' associated with F7 and F8 during the lab.

Try to explain why the debugger does not seem to treat some of the program statements as execution steps.

Draw attention to the more subtle information details displayed on the debugger MODULE view. Suggest how it may be useful.

Draw attention to the capabilities of the LOG view and their use.

Feel free to ask questions of members of the class. You may use imaginative techniques to illustrate your points.

May 28, 1990

COMPETITION #426-1120

TEMPORARY CLERK GENERAL B
FUTURES - COUNSELLING & CAREER DEVELOPMENT
MOHAWK TRAIL CAMPUS
(June 4/90 to December 24/90)

The successful applicant will provide receptionist duties for the FUTURES Program at Mohawk Trail Campus as follows:

- * maintaining operation of main switchboard
- * responding to inquiries from applicants and employers
- * assisting in screening of walk-in applicants
- * performing general office duties such as:
 - supplementary typing as required
 - distributing mail

Experience on a telephone switchboard would be an asset.

This is a temporary replacement of a permanent position. Duties will commence June 4, 1990.

Salary range is \$11.13 per hour to \$12.33 per hour (Payband 3).

All candidates are to submit a full, updated resume or application form, referring to Competition #426, by June 1, 1990, to:

Manager, Employment Services
Human Resources Division
Mohawk College of Applied Arts
and Technology
P.O. Box 2034, Hamilton,
Ontario L8N 3T2.



We are an Equal Opportunity Employer

ERIC Clearinghouse for
Junior College OCT 26 1990